The Power of Crystal:

A language for humans and computers

Johannes Müller – Crystal Core Team / Manas.Tech

Agenda

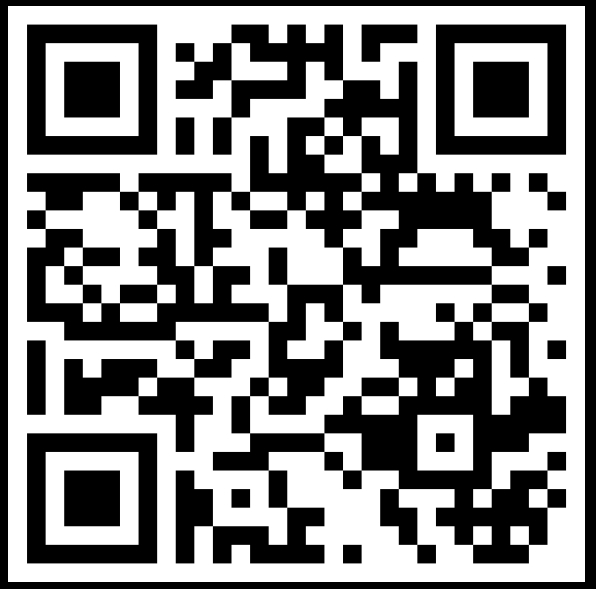
- Intro to Crystal
- Comparison with Ruby
- Combined Power
- Learnings for Ruby devs

Johannes Müller
Manas.Tech

jmuller@manas.tech

github: @straight-shoota

mastodon: @straightshoota



Follow the slides
straight-shoota.github.io/power-of-crystal

---

crystal-lang.org/install | play.crystal-lang.org

Manas.Tech

Benefits of Ruby

- User-friendly syntax
- Versatile and productive
- Simple and intuitive

Matz (1993): *“What would a **really good** programming language look like?”*

Matz (1993): *“What would a **really good** programming language look like?”*

```
puts "Hello, Ruby!"
```

Matz (1993): *“What would a **really good** programming language look like?”*

```
puts "Hello, Ruby!"
```

Ary (2011): *“What if Matz's good programming language could somehow statically compile?”*

Matz (1993): *“What would a **really good** programming language look like?”*

```
puts "Hello, Ruby!"
```

Ary (2011): *“What if Matz's good programming language could somehow statically compile?”*

```
puts "Hello, Crystal!"
```

Ruby Drawbacks

- Type checking
- Raw performance
- Software distribution
- Concurrency

Crystal's answers

- Goodies from Ruby's lineage
- Static typing with type inference
- Compiles to highly efficient machine code
- Strong, intuitive concurrency model

Syntax

```
# src/hello.crb

# Polyglot program that is valid Ruby and Crystal
# and can tell one from the other

LANGUAGE = Array.to_s == "Array(T)" ? "Crystal" : "Ruby"

puts "Hello, #{LANGUAGE}!"
```

Syntax

```
# src/hello.crb

# Polyglot program that is valid Ruby and Crystal
# and can tell one from the other

LANGUAGE = Array.to_s == "Array(T)" ? "Crystal" : "Ruby"

puts "Hello, #{LANGUAGE}!"
```

```
$ ruby src/hello.crb
Hello, Ruby!

$ crystal src/hello.crb
Hello, Crystal!
```

Batteries included

```
# A very basic HTTP server
require "http/server"

server = HTTP::Server.new do |context|
  context.response.content_type = "text/plain"
  context.response.print "Hello, Crystal!"
end

address = server.bind_tcp(8080)
puts "Listening on http://#{address}"

server.listen
```

```
$ crystal build --release src/http-server.cr

$ ./http-server &

$ wrk -t8 -c400 -d60s http://localhost:8080/
Running 1m test @ http://localhost:8080/
  8 threads and 400 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     7.11ms    1.09ms  17.66ms   79.34%
    Req/Sec     7.07k     1.15k   62.53k    85.64%
  3373196 requests in 1.00m, 334.56MB read
Requests/sec:  56126.81
Transfer/sec:      5.57MB
```

Static Typing

- Type safety
- Feels dynamic

```
def add(a, b)
  a + b
end

add 1, 2 # => 3
add "foo", "bar" => "foobar"
```

Static Typing

- Type safety
- Feels dynamic

```
def add(a, b)
  a + b
end

add 1, 2 # => 3
add "foo", "bar" => "foobar"

add "foo", 2 # Error: instantiating 'add(String, Int32)
             # Error: expected argument #1 to 'String#+'
             # to be Char or String, not Int32
```

Static Typing

```
ary = [1, 2, 3]

ary.class # => Array(Int32)

ary << 4
typeof(ary[0]) # Int32

ary << "foo" # Error: expected argument #1 to 'Array(Int32)#<<'
             # to be Int32, not String
```

Static Typing

```
ary = [1, 2, 3] of Int32 | String

ary.class # => Array(Int32 | String)

ary << 4
typeof(ary[0]) # Int32 | String

ary << "foo"
```

Static Typing

```
[] # Error: for empty arrays use '[] of ElementType'

[] of String
# or
Array(String).new
```

Static Typing

instance variables

```
class Foo
  @bar = 123

  def initialize(@baz : String)
  end
end
```

Compilation

```
$ crystal build src/hello.crb

$ ls -sh hello
2,0M hello-world

$ file hello
hello: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=d7768fdb368d11e8e08c9bcaea7cc1a629914f04, for GNU/
Linux 3.2.0, with debug_info, not stripped

$ ./hello
Hello, Crystal!
```

Compilation

- Runtime is embedded into the binary
- LLVM backend generates efficient code
- Limited dynamic language features

Metaprogramming: `macro`

Code that writes other code at compile time

Metaprogramming: `macro`

Code that writes other code at compile time

```
macro getter(var)

  # macro body

end

getter foo
```

Metaprogramming: `macro`

Code that writes other code at compile time

```
macro getter(var)

  # macro body

end

getter foo
```

Macro expansion:

```

  # macro body

```

Metaprogramming: `macro`

Code that writes other code at compile time

```
macro getter(var)
  def {{ var.id }}
    @{{ var.id }}
  end
end

getter foo
```

Macro expansion:

```
def foo
  @foo
end
```

Metaprogramming: `def`

```
class Foo
  def ==(other)
    {% for ivar in @type.instance_vars %}
      return false unless @{{ivar.id}} == other.@{{ivar.id}}
    {% end %}
    true
  end
end
```

Metaprogramming: `def`

```
class Foo
  def ==(other)
    {% for ivar in @type.instance_vars %}
      return false unless @{{ivar.id}} == other.@{{ivar.id}}
    {% end %}
    true
  end
end
```

Macro expansion:

Metaprogramming: `def`

```
class Foo
  def ==(other)
    {% for ivar in @type.instance_vars %}
      return false unless @{{ivar.id}} == other.@{{ivar.id}}
    {% end %}
    true
  end
end
```

Macro expansion:

```
# macro expansion of method body with ivars @bar and @baz
    return false unless @baz == other.@baz
    return false unless @bar == other.@bar
    true
```

API gotchas

API gotchas

- `#includes?` instead of `#include?`

API gotchas

- `#includes?` instead of `#include?`
- Only `#reduce`, no `#inject`
- Only `#size`, no `#length`

Concurrency

- Lightweight threads, CSP-style
- `spawn foo()` launches a `Fiber`
- Deeply integrated into the runtime
- `socket.puts "ping"`
- Communication via `Channel`

Dependencies

```
# shard.yml
name: my-first-crystal-app
version: 1.0.0

dependencies:
  mysql:
    github: crystal-lang/crystal-mysql
    version: >=0.16.0
```

```
$ shards install
$ shards update
```

Ruby with Crystal

Embed Crystal code directly in Ruby

```ruby
require 'crystalruby'

module MyTestModule
  # The below method will be replaced by a compiled Crystal version
  # linked using FFI.
  crystalize [a: :int, b: :int] => :int
  def add(a, b)
    a + b
  end
end

# This method is run in Crystal, not Ruby!
MyTestModule.add(1, 2) # => 3
```

wouterken/crystalruby

Crystal with Ruby

Embed an mruby interpreter in Crystal

```
require "anyolite"

Anyolite::RbInterpreter.create do |rb|
  rb.execute_script_line(%[puts "Hello from Ruby"])
end
```

Anyolite/anyolite

Combine Ruby & Crystal

- delegate performance-critical workloads
- background job processing
- service backend

Learnings for Rubyists

avoiding repeat calculations

```
abstract def bar : String | Nil
abstract def foo(arg : String)
```

```
if bar
  foo(bar) # Error: expected argument #1 to 'foo'
           # to be String, not (String | Nil)
end
```

Learnings for Rubyists

avoiding repeat calculations

```
abstract def bar : String | Nil
abstract def foo(arg : String)
```

```
if bar
  foo(bar) # Error: expected argument #1 to 'foo'
           # to be String, not (String | Nil)
end
```

```
if b = bar
  foo(b)
end
```

Learnings for Rubyists

```
{
  "action" => "foo",
  "credentials" => {
    "id" => "abc",
    "secret" => "psst"
  },
  "value" => 12,
}
```

- Language introduction tutorial
- Crystal for Rubyists
- Crystal for Rubyists
- Crystal-Shards-for-Ruby-Gems
- Excercism Track

Conclusion

- Crystal can be a useful asset in your toolbox
- Enhance your Ruby project
- Knowing Ruby makes you almost a Crystal developer
- Knowing Crystal makes you a better Ruby developer

Thank you